Café de la Neuroinformatique
iCONICS / Bioinformatic / Biostatistic Plateforme

Processing high-throughput sequencing data with a Python workflow management system: pipelines and applications
Biological basis

What is DNA?

Cell scale
Organisation in chromosomes

Body scale
Code for functional proteins
NGS technologies

What to do from genomics material?

Millions of reads to be interpreted by bioinformatics

Reads: short pieces of DNA that is sequenced by sequencers (~50 to 300 bases)

Adapted from Shyr and Liu, Biological Procedures Online, 2013, 15:4
WES, Whole Exome Sequencing

What is it?
WES, Whole Exome Sequencing

Why look in?

(Watermelon = Genome)
(Slice = Exome 1-2%)
(Seeds = Genes)

Genes alterations

(Genotype)
(Phenotype)
Problems with proteins functionalities?

DISEASES !?
Mechanisms etc...

share.ambrygen.com
WES, Whole Exome Sequencing

Testing phenotype against genotype

Familial study

Cases / Control study

Are there specific genotypes of affected individuals / population?
WES, Whole Exome Sequencing

How to identify genotype?

Sequencing PF (Yannick Marie)

- Genomic DNA
- Exome Capture
- Library construction
- Sequencing

Millions of raw reads

iCONICS PF

- QC of Raw reads
- Clean reads
- Bioinformatic analysis

- Alignment on a reference genome
- Probability of having a variation → genotype?

Variation calling
WES, Whole Exome Sequencing

Whole process summary

Steps 1 to 3:
Sequencing PF (Yannick MARIE)

10 samples ~ 48h
80 files ~ 100Go

Steps 3 to 5:
iCONICS PF

10 samples ~ 10h (ICM cluster)
During process ~ 500Go
Final ~ 100Go

share.ambrygen.com
Bioinformatic process

Workflow management system

Requirements

- Need to be efficient and fast for result rendering
  - Automatisation
  - Reproducibility
  - Portability on multiple systems with environment
  - Deployable on multiple cluster systems
WHY WORKFLOW MANAGEMENT?

Genomic analyses entail the application of various tools, algorithms and scripts.
WHY WORKFLOW MANAGEMENT?

Genomic analyses entail the application of various tools, algorithms and scripts.

Workflow management handles:

- parallelization
- suspend/resume
- logging
- data provenance
WHICH WORKFLOW MANAGEMENT?

- Galaxy
- BPIPE
- nextflow
- Cluster Flow
- LOOM
- RUBRA
WHICH WORKFLOW MANAGEMENT?

- Galaxy
- BPIPE
- nextflow
- Cluster Flow
- RUBRA
- LOOM

WHAT ABOUT SNAKEMAKE?
Python + GNU Makefile = Snakemake
A *pipeline* problem

Sample 1

- trimmomatic

- bwa_mem_after_trim

- bam_sort

- merge_bam

- dedupe

- gatk_recalibrate_info

- gatk_recalibrate_bam

- report
Ideal for embarrassingly parallel problem

Sample 1

trimmomatic

bwa_mem_aftertrim

bwa_mem_sort

merge_bam

dedupe

\texttt{gatk\_recalibrate\_info}

\texttt{gatk\_recalibrate\_bam}

report
Ideal for embarassingly parallel problem

Sample 1
  trimmomatic
  bwa_mem_after_trim
  bam_sort
  merge_bam
  dedupe
  gatk_recalibrate_info
  gatk_recalibrate_bam

Sample 2
  trimmomatic
  bwa_mem_after_trim
  bam_sort
  merge_bam
  dedupe
  gatk_recalibrate_info
  gatk_recalibrate_bam

Sample 3
  trimmomatic
  bwa_mem_after_trim
  bam_sort
  merge_bam
  dedupe
  gatk_recalibrate_info
  gatk_recalibrate_bam

Sample N
  trimmomatic
  bwa_mem_after_trim
  bam_sort
  merge_bam
  dedupe
  gatk_recalibrate_info
  gatk_recalibrate_bam

report
Why is it successful?

- Clusters can be used with minimum efforts
- Workflows can be run from or up to a given step
- Data provenance
- Nice logging system to follow the status
- Suspend / Resume
- Various code can be integrated: R, bash, and of course Python
From sequential commands to dependent rule graph:
A toy example
SNAKEMAKE - WORKFLOW

inputFile 1: sampleA
  ↓
alignment
  ↓
sort
  ↓
report

inputFile 2: sampleB
  ↓
alignment
  ↓
sort
  ↓
report
# Alignment
bwa mem genome.fa A.fastq.gz > A.bam
bwa mem genome.fa B.fastq.gz > B.bam

# Sort
samtools sort A.bam > A.sort.bam
Samtools sort B.bam > B.sort.bam

#Report
./report.sh A.sort.bam B.sort.bam
Shell command, pretty simple

Command line

Shell script

Snakefile

# Alignment
bwa mem genome.fa A.fastq.gz > A.bam
bwa mem genome.fa B.fastq.gz > B.bam

# Sort
samtools sort A.bam > A.sort.bam
Samtools sort B.bam > B.sort.bam

# Report
./report.sh A.sort.bam B.sort.bam

ISSUES
Good start. Simple but what about variables and scalability?
Shell command, pretty simple

```bash
#!/bin/sh
REFERENCE="genome.fa"
SAMPLES='ls *fastq.gz'

# Alignment
for i in $SAMPLES
do
   TARGET=${i/.fastq.gz/.bam}
   OUT=${i/.fastq.gz/.sort.bam}
   bwa mem $REFERENCE $i > $TARGET
   samtools sort $TARGET > $OUT
done

#Report
./report.sh A.sort.bam B.sort.bam
```
Shell command, pretty simple

```bash
#!/bin/sh
REFERENCE="genome.fa"
SAMPLES='ls *fastq.gz'

# Alignment
for i in $SAMPLES
do
  TARGET=${i/.fastq.gz/.bam}
  OUT=${i/.fastq.gz/.sort.bam}
  bwa mem $REFERENCE $i > $TARGET
  samtools sort $TARGET > $OUT
done

#Report
./report.sh A.sort.bam B.sort.bam
```

ISSUES
Still simple but sequential. What about dependencies between tasks? What if a file exists already? Do we start from scratch?
SAMPLES = ['sampleA', 'sampleB']

rule mapping:
  input:
    ref = 'genome.fa',
    fastq = '{sample}.fastq.gz'
  output:
    '{sample}.bam'
  shell:
    bwa mem {input.ref} {input.fastq} > {output}

rule sort:
  input:
    '{sample}.bam'
  output:
    '{sample}.sort.bam'
  shell:
    samtools sort {input} > {output}
Snakefile

```python
SAMPLES = [‘sampleA’, ‘sampleB’]

rule all:
    expand({sample}.sort.bam, sample=SAMPLES)

rule mapping:
    input:
        ref = ‘genome.fa’,
        fastq = ‘{sample}.fastq.gz’
    output:
        ‘{sample}.bam’
    shell:
        bwa mem {input.ref} {input.fastq} > {output}

rule sort:
    input:
        ‘{sample}.bam’
    output:
        ‘{sample}.sort.bam’
    shell:
        samtools sort {input} > {output}
```
- Decompose workflow into *rules*
- Rules define how to obtain output files from input files
- Infers dependencies and execution order
- How to launch your pipeline?
  - Just type this in a terminal:
  ```
snakemake -s Snakefile
  ```
rule sort:
  input:
    '{sample}.bam'
  output:
    '{sample}.sort.bam'
  threads:
    2
  resources:
    mem=8
  shell:
    samtools sort -@ {threads} {input} > {output}

snakemake -s Snakefile --cores 4
SNAKEMAKE – HOW TO LAUNCH ON CLUSTER?

- Don’t change anything in your code
- Just indicate your scheduler when you launch the pipeline

```bash
snakemake -s Snakefile \
  --cluster "sbatch" \
  --threads {threads} \
  --memory {resources.mem}\n  -p normal \
  --time 2:00:00"
```

Works on PBS, Torque, SGE, etc ...
SNAKEMAKE - SUMMARY

- A job is executed if
  - output file target does not exist
  - output file needed by another executed job and does not exist
  - input file newer than output file
  - input file will be updated by other job
  - execution is enforced

- No intrusive code: works on slurm, Torque, SGE ...

- Handle errors
  - If an error occurs after hours of computation, fix the error in your code or missing files, and run snakemake again. Finished jobs won’t be re-run.
SNAKEMAKE - SUMMARY

- handles temporary and protected files
- run until a given rule
- run from a given rule
- stats about run time: benchmark: run several times the rules
- any external scripts can be used (R, python, etc)
- remote files (http, ftp, google could, amazon, dropbox)
- rules may have priorities
- cluster time and memory can be fully customized
- modular: can include rules, or sub workflow
WHAT ABOUT REPRODUCIBILITY?

- Integrated package management

Create environment by workflow or by rule
  - yaml file which describes all software packages in the specified versions
WHAT ABOUT REPRODUCTIBILITY?

- Integrated package management
  
  → Create environment by workflow or by rule
  → yaml file which describes all software packages in the specified versions

- Containers

  → Combining Conda package management with containers